CD-ROM & the Macintosh Computer

A Gentle, Technical Introduction to Creating CD-ROMs
for the Apple Macintosh computer family

Abstract: This report discusses the use of CD-ROM technology with the Macintosh family of computers. It includes an overview of CD-ROM technology, a description of the High Sierra and ISO 9660 volume formats, a brief discussion of legal and localization issues, a description of Apple extensions to the ISO 9660 format, plus documentation of audio control calls and common questions asked about preparing CD-ROMs for the Macintosh.

© Copyright 1989 Apple Computer, Inc.

November 1989

Chapter 1. Beginner's Overview of CD-ROM. 1
Chapter 2. Roll Your Own CD-ROM. 2
Chapter 3. CD-ROM Technical Summary. 4
Chapter 4. High Sierra, ISO 9660 & The Macintosh. 13
Chapter 5. Legal issues. 19
Chapter 6. International Considerations. 20
Chapter 7. Apple Extensions to ISO 9660. 22
Chapter 8. Macintosh Audio Calls. 32
Chapter 9. Ask Swami ROMmie. 47
Chapter 10. Operating System Support for ISO 9660. 51

Chapter 1. Beginner's Overview of CD-ROM

CD-ROM stands for **C**ompact **D**isc - **R**ead **O**nly **M**emory. A CD-ROM is a silvery platter 120mm in diameter. CD-ROM uses the same form factor and plastic as audio CDs; in fact, they share a common definition at the bit level.

CD-ROM discs have the following characteristics:

large storage capacity	660 Megabytes per CD when dedicated to digital data.
mixed sound/data	a CD can contain both digital data and high quality audio CD tracks, thus opening up new possibilities.
stable storage	because CD-ROMs can't be overwritten, they are ideal for storing static data without fear of data loss due to operator error.
low production cost	CDs can cost less than two dollars each to produce. This is almost the same production cost as a single floppy disk which holds only 800 kilobytes. You can fit 825 800K floppies on one CD-ROM.
standard format(s)	Any CD-ROM can be read at the bit level by any CD-ROM player, just as any audio CD can be played on any audio CD player. CD-ROM even has an operating system-independent file format, known as High Sierra, which allows a single CD-ROM disc to be used on a variety of computers.

The physical format of both audio CDs and CD-ROM discs is exhaustively explained in a standard published by Philips and Sony and known as the Yellow Book. CD-ROM needs additional definitions for the error correcting schemes and so forth; these are set forth in another Philips/Sony standard known as the Red Book. Because these standards exist and are followed world-wide, you are assured that all CD-ROMs can be read in all CD-ROM players, at least at the bit level.

Reading bits is interesting, but it isn't enough. When a CD-ROM is pressed in a special volume structure known as High Sierra (ISO 9660), the CD can be read by most popular personal computer operating systems. If you use a standard application data format (such as SYLK or WKS for spreadsheets or dBase III for databases) then you can share data easily between different computer types.

If you want to take advantage of a particular operating system, or if you want to keep your data protected against outside use, you can also press CDs in the native operating system file format. For instance, since Hypercard is a product that runs only on Macintosh computers, many people who create large Hypercard stacks use the native Macintosh file system format for their CDs. It saves the hassle of doing the conversion into the ISO 9660 format.

Chapter 2. Roll Your Own CD-ROM

(Or, How to Succeed In Plastic Without Hardly Trying)

Summary: You can press 100 CDs for around $1700, within 10 days, using things you already have in your office. More CDs are $2.00 each.

You should read this paper if you're interested in pressing a CD and don't know how to do it. We'll assume that you're looking at a moderate number of CDs (one hundred to several thousand) for your needs. It makes sense to press a CD if you have 10 megabytes or more of data and need 100 or more copies of it.

CDs behave just like a locked, very fast, very big floppy. If you can run your program or Hypercard stack on a locked floppy, it will run on a CD.

To press a CD-ROM, you have two options; the easy way and the hard way. The easy way gives you a Macintosh CD that's a copy of a hard disk. The hard way gives you the Macintosh CD plus the ability to create audio CD quality sound tracks, or to create a CD that can be used across many operating systems in a format called *High Sierra*.

The easy way is to put your data onto a hard disk and send the hard disk to one of the companies listed below (call before sending). They will create an image copy of your hard disk (an exact duplicate) and put that image onto CDs. Easy, quick and painless. To create such a CD, you'll be charged a *mastering charge* (for creating the initial image) plus a *duplication charge* per CD. The mastering charge varies, but is usually under $1500. The duplication charge is around $2.00 per CD, including packaging. The average turn-around time is 5 to 10 days.

If you want faster turn-around time, you'll pay additional money. Typical mastering charges for one day turnaround starts under $3000. If you want zillions and zillions of CDs, you'll pay less money. Any of the production companies will be thrilled to hear from you.

The hard way of pressing a CD-ROM involves using a *pre-mastering system* to create a tape that a pressing plant can use to create CDs. This provides much more flexibility, since you control what goes on the tape. You need much more technical knowledge, and you need access to a pre-mastering facility. You can purchase a pre-mastering system; you've got names in the *Apple CD-ROM Development Resource Guide*. There are companies that will rent you time on their pre-mastering system and help you out with the details (for a fee).

If you want to press a CD that can be used on many different operating systems, you want to use a volume format called *High Sierra* (also known by the name of its international standard, *ISO 9660*). If you press a CD in this format, you can read the data on many different operating systems. You'll need to plan your data carefully so that it can be used on many machines; for example, many spreadsheets can read a common format, and almost every word processor can read plain text. Hypercard uses a proprietary format which can only be used on the Macintosh, so it's generally not a good choice for a High Sierra CD.

For a good overview of CD-ROM, read the *AppleCD SC Developer's Guide*, available from APDA. There is a new magazine called *CD-ROM EndUser*, available from DDRI, 510 North Washington Street, Suite 401, Falls Church, Virginia 22046.

Some companies to contact regarding CD-ROM production (in no particular order):

3M Optical Recording
Building 223-5S-01 / 3M Center
St. Paul, MN 55144
612/736-3274
Mark Arps/Dick Tendill
AppleLink: D2462

DADC
1800 N. Fruitridge Avenue
Terre Haute, IN 47804
812/462-8100
Linda Watson/Kozo Arai
AppleLink: D2125

Disctronics
1120 Cosby Way
Anaheim, CA 92806
714/630-6700
Wan Seegmiller

Philips Dupont Optical
1409 Foulk Road, Suite 200
Wilmington, DE 19803
800/433-3475
Jill Jones
AppleLink: D2173

Nimbus Information Systems
SR 629, Guildford Farm
Ruckersville, VA 22968
800/782-0778
Larry Boden

Denon America
222 New Road
Parsippany, NJ 07054
201/575-2532
Nob Tokutake/Ben Garcia

Some technical hints for making your CD run fast:

- Don't put 1000 files in one folder. The Finder is very slow at opening such folders.
- Make your disk contiguous for speed. Either backup, reformat the disk and copy everything on again, or use a commercial disk organizer such as DiskExpress by AlSoft or SUM by Symantec.
- Waste disk instead of cycles. Precompute information that speeds up your program or stack; you've got 660 Megabytes to waste. Use them.
- Take advantage of the static nature of CD-ROM. Put absolute file names in your Hypercard stacks.

(The legal beagles want me to tell you: All company names and prices are here for your information, but Apple Computer, Inc. does not recommend or endorse any particular company listed. Apple, HyperCard and Macintosh are registered trademarks of Apple Computer, Inc. AppleCD SC and APDA are trademarks of Apple Computer, Inc. DiskExpress is a trademark of AlSoft, Inc. SUM is a trademark of Symantec, Inc.)

Chapter 3. CD-ROM Technical Summary

From Plastic Pits to "Fantasia"
March, 1988

Abstract

This summary describes how information is encoded on Compact Disc (CD) beginning with the physical pits and going up through higher levels of data encoding to the structured multimedia information that is possible with programs like HyperCard. This discussion is much broader than any single standards document, e.g. the *CD-Audio Red Book* , while omitting much of the detail needed only by drive manufacturers.

Salient Characteristics

1. High information density -- With the density achievable using optical encoding, the CD can contain some 540 megabytes of data on a disc less than five inches in diameter.

2. Low unit cost -- Because CDs are manufactured by a well-developed process similar to that used to stamp out LP records, unit cost in large quantities is now about two dollars.

3. Read only medium -- CD-ROM is read only; it cannot be written on or erased. It is an electronic publishing, distribution, and access medium; it cannot replace magnetic disks.

4. Modest random access performance -- Due to optical read head mass and data encoding methods, random access ("seek time") performance of CD is better than floppies but not as good as magnetic hard disks.

5. Robust, removable medium -- The CD itself is comprised mostly of, and completely coated by, durable plastic. This fact and the data encoding method allow the CD to be resistant to scratches and other handling damage. Media lifetime is expected to be long, well beyond that of magnetic media such as tape. In addition, the optical servo scanning mechanism allows CDs to be removed from their drives.

6. Multimedia storage -- Because all CD data is stored digitally, it is inherently multimedia in that it can store text, images, graphics, sound, and any other information expressed in digital form. Its only limit in this area is the rate at which data can be read from the disc, currently about 150 KBytes/second. This is sufficient for all but uncompressed, full motion color video.

CD Data Hierarchy

Storing data on a CD may be thought of as occurring through a data encoding hierarchy with each level built upon the previous one. At the lowest level, data is physically stored as pits on the disc. It is actually encoded by several low-level mechanisms to provide high storage density and reliable data recovery. At the next level, it organized into tracks which may be digital audio or CD-ROM. The High Sierra specification then defines a file system built on CD-ROM tracks. Finally, applications like HyperCard specify a content format for files.

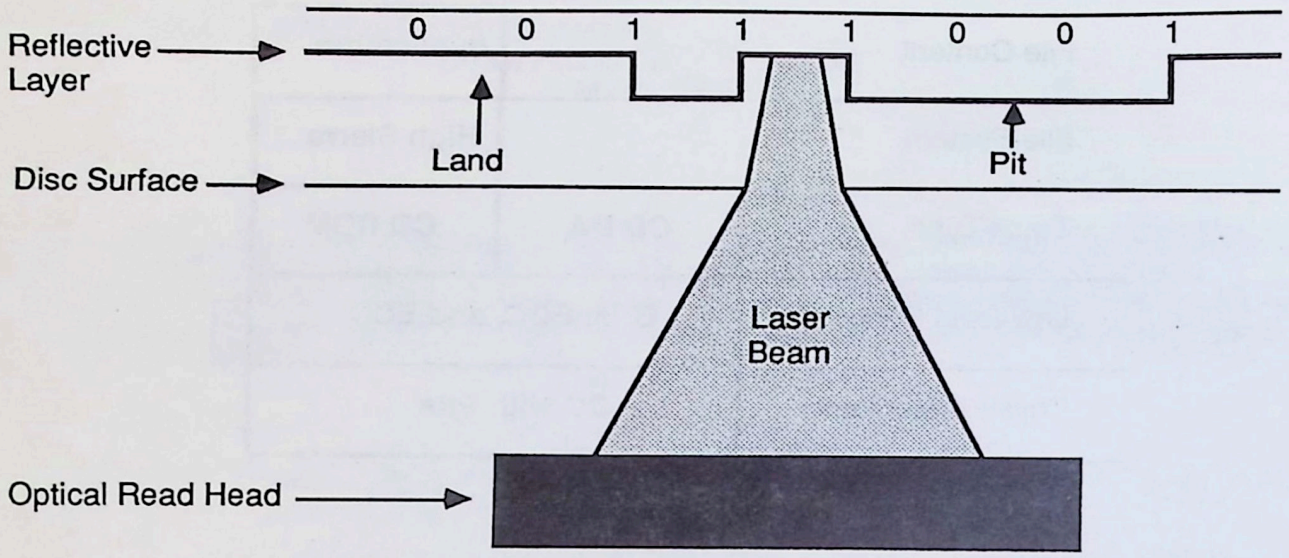
File Content		**Application**
File System		**High Sierra**
Track Type	**CD DA**	**CD ROM**
Low-level Encoding	**EFM, EDC, and ECC**	
Physical Medium	**CD with Pits**	

CD Data Hierarchy

The Physical Medium

The Compact Disc itself is a thin plastic disk some 12 cm. in diameter. Information is encoded in a plastic-encased spiral track contained on one side of the disk. The spiral track is read optically by a noncontact head which scans approximately radially as the disk spins just above it. The spiral is scanned at a constant linear velocity thus assuring a constant data rate. This requires the disc to rotate at a decreasing rate as the spiral is scanned from its beginning near the center of the disc to its end near the disc circumference.

The spiral track contains shallow depressions, called pits, in a reflective layer. Binary information is encoded by the lengths of these pits and the lengths of the areas between them, called land. During reading, a low power laser beam from the optical head is focused on the spiral layer and is reflected back into the head. Due to the optical characteristics of the plastic disc and the wavelength of light used, the quantity of reflected light varies depending on whether the beam is on land or on a pit. The modulated, reflected light is converted to a radio frequency, raw data signal by a photodetector in the optical head.



Compact Disc Cross Section

Low-level Data Encoding

To ensure accurate recovery, the disc data must be encoded to optimize the analog-to-digital conversion process that the radio frequency signal must undergo. Goals of the low level data encoding include:

1. High information density. This requires encoding that makes the best possible use of the high, but limited, resolution of the laser beam and read head optics.

2. Minimum intersymbol interference. This requires making the minimum run length, i.e. the minimum number of consecutive zero bits or one bits, as large as possible.

3. Self-clocking. To avoid a separate timing track, the data should be encoded so as to allow the clock signal to be regenerated from the data signal. This requires limiting the maximum run length of the data so that data transitions will regenerate the clock.

4. Low digital sum value (the number of one bits minus the number of zero bits). This minimizes the low frequency and DC content of the data signal which permits optimal servo system operation.

A straightforward encoding would be to simply to encode zero bits as land and one bits as pits. However, this does not meet goal (1) as well as the encoding scheme actually used. The current CD scheme encodes one bits as transitions from pit to land or land to pit and zero bits as constant pit or constant land.

To meet goals (2) to (4), it is not possible to encode arbitrary binary data. For example, the integer 0 expressed as thirty-two bits of zero would have too long a run length to satisfy goal (3). To accommodate these goals, each eight-bit byte of actual data is encoded as fourteen bits of channel data. There are many more combinations of fourteen bits (16,384) than there are of eight bits (256). To encode the eight-bit combinations, 256 combinations of fourteen bits are chosen that meet the goals. This encoding is referred to as Eight-to-Fourteen Modulation (EFM) coding.

If fourteen channel bits were concatenated with another set of fourteen channel bits, once again the above goals may not be met. To avoid this possibility, three merging bits are included between each set of fourteen channel bits. These merging bits carry no information but are chosen to limit run length, keep data signal DC content low, etc. Thus, an eight bit byte of actual data is encoded into a total of seventeen channel bits: fourteen EFM bits and three merging bits.

To achieve a reliable self-clocking system, periodic synchronization is necessary. Thus, data is broken up into individual frames each beginning with a synchronization pattern. Each frame also contains twenty-four data bytes, eight error correction bytes, a control and display byte (carrying the subcoding channels), and merging bits separating them all. Each frame is arranged as follows:

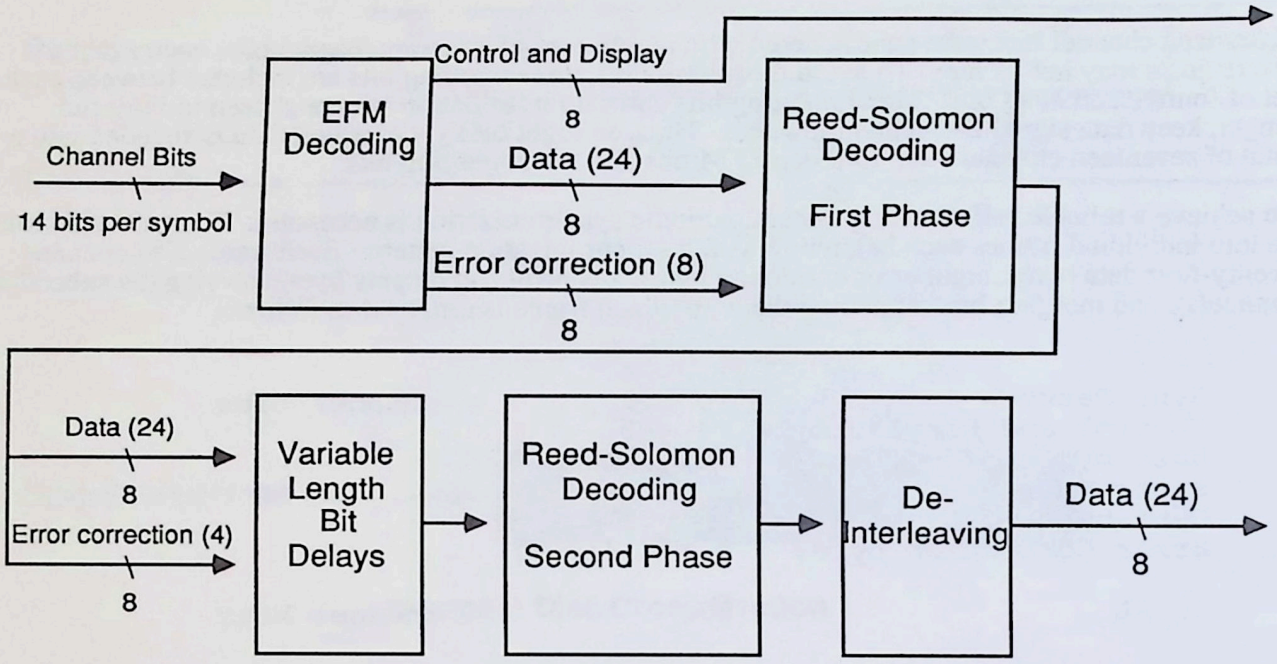
Sync Pattern	24 + 3	channel bits
Control and Display byte	14 + 3	
Data bytes	12 * (14 + 3)	
Error Correction bytes	4 * (14 + 3)	
Data bytes	12 * (14 + 3)	
Error Correction bytes	4 * (14 + 3)	
TOTAL	588	channel bits

Thus, 192 actual data bits (24 bytes) are encoded as 588 channel bits.

First Level Error Correction

Data errors can arise from production defects in the disk itself, defects arising from subsequent damage to the disk, or jarring during reading. A significant characteristic of these errors is that they often occur in long bursts. This could be due, for example, to a relatively wide mark on the disc that is opaque to the laser beam used to read the disc. A system with two logical components called the Cross Interleave Reed-Solomon Coding (CIRC) is employed for error correction. The cross interleave component breaks up the long error bursts into many short errors; the Reed-Solomon component provides the error correction.

As each frame is read from the disc, it is first decoded from fourteen channel bits (the three merging bits are ignored) into eight-bit data bytes. Then, the bytes from each frame (twenty-four data bytes and eight error correction bytes) are passed to the first Reed-Solomon decoder which uses four of the error correction bytes and is able to correct one byte in error out of the 32. If there are no uncorrectable errors, the data is simply passed along. If there are errors, the data is marked as being in error at this stage of decoding.



Compact Disc Data Decoding

The twenty-four data bytes and four remaining error correction bytes are then passed through unequal delays before going through another Reed-Solomon decoder. These unequal delays result in an interleaving of the data that spreads long error bursts among many different passes through the second decoder. The delays are such that error bursts up to 450 bytes long can be completely corrected. The second Reed-Solomon decoder uses the last four error correction bytes to correct any remaining errors in the twenty-four data bytes. At this point, the data goes through a de-interleaving process to restore the correct byte order.

Subcoding Channels and Blocks

The eight-bit control and display byte in each frame carries the subcoding channels. A subcoding block consists of 98 subcoding bytes, and thus 98 of the 588-bit frames. A block then can contain 2352 bytes of data. Seventy-five blocks are read each second. With this information, it is now straightforward to calculate that the CD data rate is in fact correct for CD digital audio (CD-DA):

> Required CD digital audio data rate: 44.1 K samples per second * 16 bits per sample * 2 channels = 1,411,200 bits/sec.
>
> CD data rate: 8 bits per byte * 24 bytes per frame * 98 frames per subcoding block * 75 subcoding blocks per second = 1,411,200 bits/sec.

The eight subcoding channels are labeled P through W and are encoded one bit for each channel in a control and display byte. Channel P is used as a simple music track separator. Channel Q is used for control purposes and encodes information like track number, track type, and location (minute, second, and frame number). During the lead-in track of the disc, channel Q encodes a table of contents for the disk giving track number and starting location. Standards have been proposed that would use the remaining channels for line graphics and ASCII character strings, but these are seldom used.

Track Types

Tracks can have two types as specified in the control bit field of subchannel Q. The first type is CD digital audio (CD-DA) tracks. The two-channel audio is sampled at 44.1 Khz with sixteen bit linear sampling encoded as twos complement numbers. The sixteen bit samples are separated into two eight-bit bytes; the bytes from each channel alternate on the disc. Variations for audio tracks include pre-emphasis and four track recording.

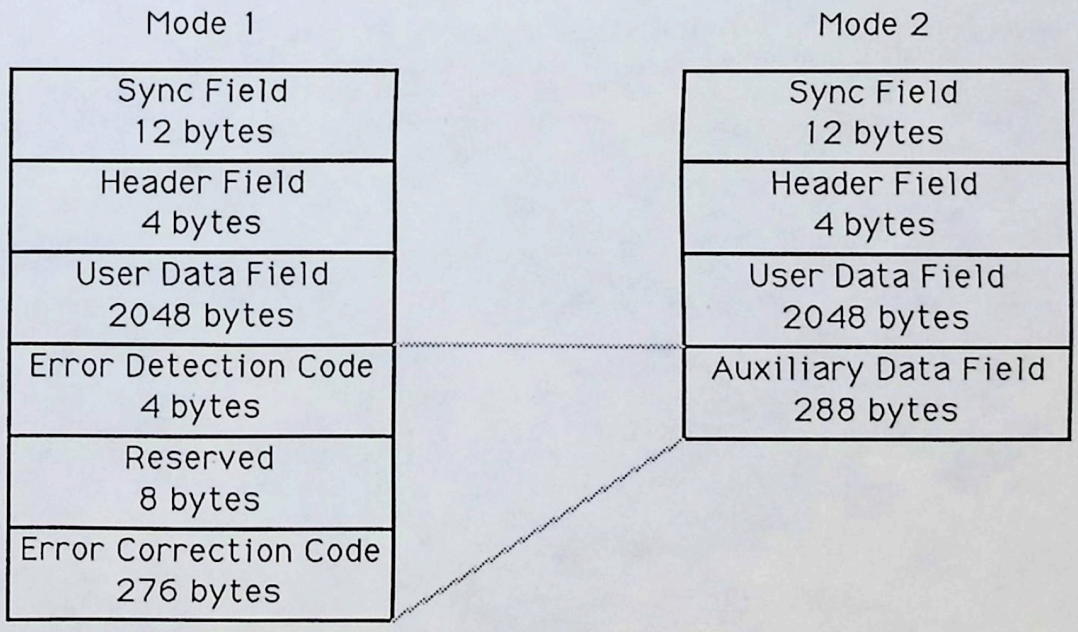
The other type of track specified by the subchannel Q control bit field is the data track. These must conform to the CD-ROM standard described below. In general, a disc can have a mix of CD digital audio tracks and a CD-ROM track, but the CD-ROM track must come first.

CD-ROM Data Tracks

Each CD-ROM data track is divided into individually addressable blocks of 2352 data bytes, i.e. one subcoding block or 98 frames. A header in each block contains the block address and the mode of the block. The block address is identical to the encoding of minute, second, and frame number in subcode channel Q. The modes defined in the CD-ROM specification are:

Mode Value	User Data Field	Auxiliary Data Field
0	all zero bytes	all zero bytes
1	data bytes	EDC/ECC bytes
2	data bytes	data bytes

Thus, mode 1 defines separately addressable, physical 2K byte data blocks making CD-ROM look at this level very similar to other digital mass storage devices.



CD-ROM Block Format

Second Level Error Correction

An uncorrected error in audio data typically results in a brief, often inaudible click during listening at worst. An uncorrected error in other kinds of data, for example program code, may render a CD unusable. For this reason, CD-ROM defines a second level of error detection and error correction (EDC/ECC) for mode 1 data. The information for the EDC/ECC occupies most of the auxiliary data field.

The error detection code is a cyclic redundancy check (CRC) on the sync , header, and user data. It occupies the first four bytes of the auxiliary data field and provides a very high probability that uncorrected errors will be detected. The error correction code is essentially the same as the first level error correction in that interleaving and Reed-Solomon coding are used. It occupies the final 276 bytes of the auxiliary data field.

The High Sierra File System Standard

Built on top of the addressable 2K blocks that the CD-ROM specification defines, the next higher level of data encoding is a file system that permits logical organization of the data on the CD. This can be a native file system like the Macintosh Hierarchical File System (HFS). Another alternative is the High Sierra (also known as the ISO 9660) file standard, recently approved by the National Information Standards Organization (NISO) and the International Standards Organization (ISO), which defines a file system carefully tuned to CD characteristics. In particular:

1. CDs have modest seek time and high capacity. As a result, the High Sierra standard makes tradeoffs that reduce the number of seeks needed to read a file at the expense of space efficiency.

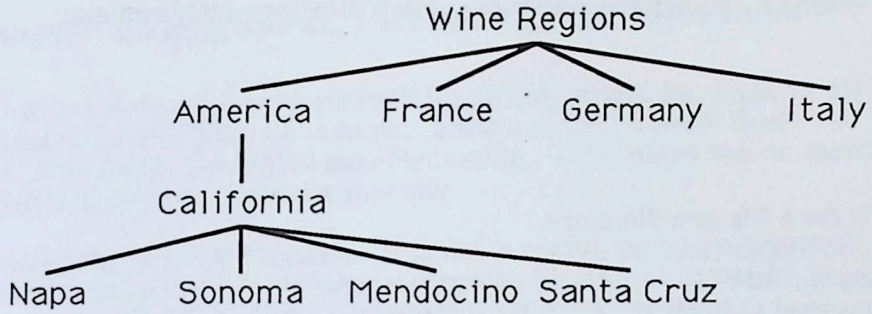
2. CDs are read-only. Thus, concerns like space allocation, file deletion, and the like are not addressed in the specification.

For High Sierra file systems, each individual CD is a volume. Several CDs may be grouped together in a volume set and there is a mechanism for subsequent volumes in a set to update preceding ones. Volumes can contain standard file structures, coded character set file structures for character encoding other than ASCII, or boot records. Boot records can contain either data or program code that may be needed by systems or applications.

High Sierra Directories and Files

The file system is a hierarchical one in which directories may contain files or other directories. Each volume has a root directory which serves as an ancestor to all other directories or files in the volume. This dictates an overall tree structure for the volume.

A typical disadvantage in hierarchical systems is that to read a file (which must be a leaf of the hierarchy tree) given its full path name, it is necessary to begin at the root directory and search through each of its ancestral directories until the entry for the file is found. For example, given the path name "Wine Regions: America: California: Mendocino", three directories (the first three components of the path name) would need to be searched. Typically, a separate seek would be required for each directory. This would result in relatively poor performance.



High Sierra Hierarchical File System

To avoid this, High Sierra specifies that each volume contain a path table in addition to its directories and files. The path table describes the directory hierarchy in a compact form that may be cached in computer memory for optimum performance. The path table contains entries for the volume's directories in a breadth-first order; directories with a common parent are listed in lexicographic order. Each entry contains only the location of the directory it describes, its name, and the location in the path table of its parent. This mechanism allows any directory to be accessed with only a single CD seek.

Directories contain more detailed information than the path table. Each directory entry contains:

- Directory or file location.
- File length.
- Date and time of creation.
- Name of the file.
- Flags:
  - Whether the entry is for a file or a directory.
  - Whether or not it is an associated file.
  - Whether or not it has records.
  - Whether or not it has read protection.
  - Whether or not it has subsequent extents.
- Interleave structure of the file.

Interleaving may be used, for example, to meet realtime requirements for multiple files whose contents must be presented simultaneously. This would happen if a file containing graphic images were interleaved with a file containing compressed sound that describes the images.

Files themselves are recorded in contiguous (or interleaved) blocks on the disc. The read-only nature of CD permits this contiguous recording in a straightforward manner. A file may also be recorded in a series of noncontiguous extents with a directory entry for each extent.

The specification does not favor any particular computer architecture. In particular all significant, multibyte numbers are recorded twice, once with the most significant byte first and once with the least significant byte first.

Multimedia Information

Using the file system are applications that create and portray multimedia information. While it is true that a CD can store anything that a magnetic disk can store (and usually much more of it), CDs will be used more for storing information than for storing programs. It is the very large storage capacity of CDs coupled with their low cost that opens up the possibilities for interactive, multimedia information to be used in a multitude of ways.

Programs like HyperCard, with it's ease of authoring and broad extensibility, are very useful for this purpose. Hypercard stacks, with related information such as color images and sound, can be easily and inexpensively stored on CDs despite their possibly very large size.

Chapter 4. High Sierra, ISO 9660 & The Macintosh

CD-ROM discs all use the same Philips/Sony format for the physical organization of the disc. This format is specified in two volumes, known as the "Yellow Book" for lowest level physical formatting, and "Red Book" for digital data formatting. This means that no matter who creates the disc, every CD-ROM drive can read the raw bits.

Having this physical format in common is useful, but it's not enough. We also need to be able to find specific files on a CD; we need a common directory format. High Sierra is a common directory format for CD-ROM discs, not matter what operating system.

What is High Sierra?

A group of industry representatives met at Del Webb's High Sierra Hotel & Casino in the mountains of the U.S. state of Nevada, in late 1985 to see if companies could cooperate in developing a common directory format for CD-ROM discs. The result of this series of meetings was a standard known as the "High Sierra" standard. This standard is fully specified by the *May 28, 1986 Working Paper for Information Processing—Volume and File Structure of Compact Read Only Optical Discs for Information Interchange*. For obvious reasons, this is known as the High Sierra paper.

Like all good standards, the world at large wanted to adopt an equivalent standard. The International Organization for Standardization, ISO, has pushed the High Sierra standard through it's standardization process. What's come out the other end is a new international standard, ISO 9660—*Volume and File Structure of CD-ROM for Information Interchange*. This is known as the ISO standard. ISO 9660 is the wave of the future—most existing CD-ROMs are High Sierra, but everyone is changing over to the ISO 9660 standard, and most future discs will be ISO 9660 standard rather than High Sierra. Apple's operating systems (Macintosh and IBM) and Microsoft's operating system MS-DOS all support both the High Sierra standard and the ISO 9660 standard. In my comments below, "High Sierra" refers to both ISO 9660 and High Sierra unless I explicitly state differently.

High Sierra has some basic design goals:
- easy implementation under a variety of operating systems, such as the Macintosh operating system, Apple II ProDOS 16 or GS/OS, MS-DOS, Unix, VMS, etc.
- optimize the design for read-only media. High Sierra is meant to be used only on CD-ROM discs.
- minimize the number of seeks to be performed on the CD, to account for the relatively slow seek time of CD-ROM drives.

Because of these design goals, the High Sierra file system is different than most popular file systems. High Sierra is a read-only file system. The file system is optimized for CD-ROM, with its slow seek times and large capacities. Because there are two basic ways of ordering 16 bit numbers in the microprocessor world, all appropriate numeric information within the system is stored in both least-significant-byte first order (lsb) and most-significant-byte first order (msb). (Motorola microprocessors such as those used in the Apple Macintosh store numeric information internally in lsb order. Intel processors such as those used in MS-DOS compatible computers store numeric information in msb order.)

The High Sierra specification describes the five parts that make up the volume. High Sierra does not specify the content of any file; it only describes how to get to the file. The five pieces that High Sierra defines are:

- The logical format of the volume. The logical format is how you describe an arbitrary location on the volume.
- The volume descriptors, which contain information about the entire contents of the volume.

- The path tables, which are an easy way to get to any directory on the volume.
- The directory and directory record formats, which describe the information that an operating system needs to know about how to get to a file.
- File information, containing what an operating system needs to access a specific file.

High Sierra (or the equivalent international standard, ISO 9660) has many support in many different operating systems. Apple Macintosh, Apple II ProDOS, Apple IIgs GS/OS, MS-DOS, and VMS all have High Sierra support available, as does many varieties of Unix.

What's the Difference between High Sierra and ISO 9660?

The differences between High Sierra and ISO 9660 are slight, and mostly of interest to programmers. For the technically interested, the differences are as follows:

- The primary and secondary volume descriptors have some differences in the type and quantity of fields.
- A bibliographic preparer field was added to the primary and secondary volume descriptors.
- You can have up to four copies of the path table in High Sierra, but only two copies in ISO 9660.
- Two fields exchanged position in the directory records.
- All date/time fields have an extra byte, used to describe the 15 minute offset from Universal Standard Time (GMT, or UTC.)

The ISO 9660 format

The ISO 9660 format is described in the paper *ISO 9660—Volume and File Structure of CD-ROM for Information Interchange*. This specification is exact, but very difficult to read. This specification is available from the International Organization for Standardization, Geneva, Switzerland.

The description that follows is a reasonably technical description of the ISO 9660 format; it is *not* the definitive description of the format. For the one true, proper definition of ISO 9660, read the original specification.

The Macintosh File System

The Apple Macintosh file system is named "HFS", for "Hierarchical File System." This file system is similar to both MS-DOS and Unix in supporting subdirectories, where files may be logically grouped together. Subdirectories on the Macintosh are called *folders*.

When the Macintosh was first introduced, a flat file system (called "MFS," for "Macintosh File System") was used. The MFS file system did not adequately provide for high capacity storage media such as hard disks and CD-ROM, so the HFS format was introduced in its place. MFS format is now generally used only on 400K, single sided floppy disks. All Macintosh computers support MFS for backward compatibility.

Macintosh file and volume names are different from both Unix and MS-DOS in that they allow any character except the colon ":" as a valid file name character. This means that file names such as "My payroll file" or "Åéîøü" are perfectly acceptable on the Macintosh. There is no concept of a file name extension, such as is found in the MS-DOS or High Sierra formats. HFS does not distinguish between upper and lower case names; the names "forecast", "Forecast", and "FoReCaSt" all refer to the same file. Volume names may have a maximum of 27 characters, and file names may have a maximum of 31 characters. To specify a file name in a folder (subdirectory), you give the folder name, followed by a colon, followed by the file name. For example, if you have the file "My payroll file" in the folder "business", you could specify the file as "business:My payroll file". To specify a file in the folder immediately above the current folder, you may use the shortcut name of two colons, "::".

An area where HFS is different from MS-DOS or Unix is in the concept of file forks. A file has two forks; a resource fork and a data fork. The data fork is used for the same purpose as a "file" in MS-DOS or Unix; it contains information in an unstructured format. The data fork is used by an application to store the contents of the document.

The resource fork of a file contains Macintosh resources. Resources are data in a special format; Apple Computer has defined many "famous" types of resources, and we have provided facilities so that programmers may define their own custom resource types. The resource fork of a file is accessed using special program calls, from a part of the Macintosh operating system known as the Resource Manager. All resources are labeled with a resource type of four characters. Some examples of resources are: 'CODE', for executable code; 'ICON' for a 16 bit by 16 bit icon; 'ICN#', a slightly different icon resource used by the Finder to display an unique icon for a file; 'MENU' for the menus found at the top of every Macintosh application; and 'STR ', for strings used within a program. There are many other types of resources as well.

Resources provide the Macintosh with a flexibility not found in other operating systems. For instance, to "localize" a well-written application into another language, all that needs to be changed are the resources that are language specific (usually the 'MENU' and 'STR ' resources.) Apple provides developers with several tools and resource editors that assist in modifying resources.

As well as the resource and data forks, additional information is kept about each file. This information is used by the Finder to display the position, attributes, and icon of a file. The additional information consists of a file type of four characters (such as 'TEXT' for plain text or unknown type documents, 'APPL' for applications, 'MACA' for MacWrite documents, and so forth), a file creator unique to each application (such as 'MWRT', 'EXCL', 'FNDR') and file attributes (such as invisible, system file, or locked against accidental deletion.)

The additional information also includes a mapping of file creator to appropriate application, so that a Macintosh user may select a document and open it, and the appropriate application will be invoked and passed the document name.

For further information about the Macintosh file system, see the appropriate chapters in *Inside Macintosh*, the standard reference book for programming on the Macintosh. Also see *Technical Introduction to the Macintosh Family* and *Programmer's Introduction to the Macintosh Family*, both available from Addison-Wesley.

Logical Format

ISO 9660 describes everything in terms of *logical sectors* and *logical blocks*. Logical sectors are always 2048 bytes long. Logical blocks can be either 512 bytes, 1024 bytes, or 2048 bytes long. Both logical sectors and logical blocks start at 1, and are continuous.

Volume Descriptors

Volume descriptors describe information used for the entire volume. There are five types of volume descriptors: primary volume descriptor, secondary volume descriptor, boot descriptor, partition descriptor, and volume descriptor terminator. Every volume descriptor is one logical sector long (i.e. 2048 bytes.) The volume descriptors start at logical sector 16 on the disk.

You can think of the volume descriptors as an array of 2048 byte entries, beginning at sector 16 on the disk, with the first descriptor always a primary volume descriptor, and the last descriptor always a volume descriptor terminator.

There is always one and only one *primary volume descriptor* on a volume. This primary volume descriptor starts at logical sector 16 on the CD. The primary volume descriptor contains the volume name, some publishing information, and offsets to the other important information on the disk such as the path table and root directory. The primary volume descriptor also contains a copy of the root directory entry (to minimize the number of seeks necessary to find out information about a disc.) All characters in a primary volume descriptor are in the range "A" to "Z" (in English) and the digits "0" through "9". This is a subset of ISO 646, the standard for 7 bit character representation.

You may have zero, one, or more than one *secondary volume descriptor* on a volume. The secondary volume descriptor contains the same information as the primary volume descriptor, but the characters used to describe the information aren't restricted to ISO 646. These secondary volume descriptors allow you to press a CD-ROM so that it can display the directories in a non-Roman character set such as Japanese, Hebrew, or Arabic.

The secondary volume descriptors contain pointers to the equivalent information about files as the primary volume descriptor; they are just in a different character set. You can think of secondary volume descriptors as a parallel, shadow, directory structure in another alphabet.

A *boot descriptor* contains information for a particular operating system, enabling that operating system to boot from the CD-ROM. You may have more than one boot descriptor per volume, to accommodate several different operating systems.

The *volume descriptor terminator* is a very simple structure. There is only one volume descriptor terminator per volume. It servers to indicate that you have reached the end of the volume descriptor array.

Path Tables

The *path table* was designed to minimize the number of seeks necessary to get to a file's directory information. The path table contains a list of every directory in the volume, along with information about each directory. Each path table record contains the offset of the directory, the path table record number of the parent of this directory (discussed below), the name of the directory, and the size of the extended attribute directory (also discussed below.) With this information, you potentially need only one seek to get to the file information.

Directories

Directories are stored in a hierarchical tree. There is a parent, root, directory. All other directories are descended from this root. You can nest subdirectories up to eight levels deep (the root plus seven levels.)

Directory Records are the basic unit of information kept about each file. Each directory records contains the offset to the file itself, the size of the file, date and time information for creation and modification, file attribute flags, information useful for interleaved files (defined below), and the file name identifier. There is also an optional extension field, which is used by the Apple Macintosh and Apple II operating systems to store additional information not defined by the High Sierra format which is necessary in the operating system.

Additional file information necessary for multiuser operating systems such as Unix or VMS is retained in the extended attribute record.

File identifier consist of the upper case alphabet, numbers, and the underscore character '_'. File identifiers consist of a file name, a period, a file extension, a semicolon, and a file version number. File identifiers can be up to 31 characters long, but there is a "level 1 conformance," designed for

compatibility with MS-DOS, which restricts file names to eight characters, a period, and three characters.

There are two types of files: regular files, and associated files. Regular files are the same as the traditional file concept used in an operating system such as Unix or MS-DOS. A regular file without an associated file is simply a stream of bytes.

An associated file is a file with the same name as a regular file, and with the associated file attribute bit set. This file is used on the Macintosh to store the resource fork of a Macintosh file. The data fork of a Macintosh file is stored as the regular file corresponding to the associated file.

Using ISO/High Sierra on the Macintosh

The Apple IIgs and Apple Macintosh both support the 28 May 1986 "High Sierra" format and the international standard ISO 9660 format.

Apple supports both High Sierra formats through the use of a feature in the Macintosh file system called the "external file system" hook. This is a special low-memory global which contains a pointer to a list of external file system handlers. For High Sierra support, Apple has written a new set of routines contained in a file called "Foreign File Access". This file, plus the files "High Sierra File Access" and "ISO 9660 File Access" combine to provide complete support for the High Sierra formats. Apple's High Sierra software supports level 2 interchange, according to section 10.2 of the ISO 9660 specification. This means that Apple does not yet support interleaved files or multi-disc volumes.

The High Sierra format is meant to be operating system independent. HFS maps reasonably well onto the High Sierra format. A file's data fork is equivalent to a regular file in the High Sierra format. A file's resource fork is equivalent to an associated file. The additional information about each file that is needed by the Finder is not specified in either 28 May 1986 High Sierra nor in ISO 9660. To accommodate this information, Apple Computer has defined an extension to the ISO 9660 standard, using the "System Use" field defined in section 9.1.13 of the ISO 9660 specification. Please see the chapter *Apple Extensions to ISO 9660* for a complete specification of the extensions.

The Apple extensions do not fully correct deficiencies in the High Sierra specification; some issues are handled by the High Sierra File Access software. For instance, in HFS, the Finder keeps track of the position of a file on the desktop or in a folder by using a special field inside HFS; under High Sierra, a file's position is computed when the folder is opened, and can not be changed.

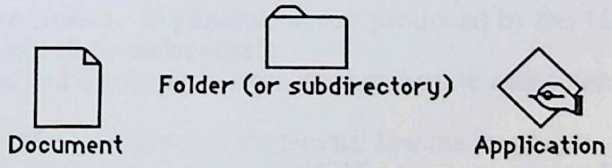


Figure 1: Generic document and generic application icon, plus standard folder icon.

Because of some deficiencies in the original design of the Finder, it is not possible to have the correct icons display for a file on a High Sierra disc. This is because the Finder does not actually ask for the icon of a file; rather, it assumes the existence of a *desktop database* which contains these mappings, and makes a special call, giving only the file creator and type. The software to provide this information was designed to be very HFS specific. Currently, even if a file's icon bitmap is defined in the Apple extensions, it will not be used by the Finder. All files that reside on a High Sierra disc will display a generic icon. If such a file is copied to a hard disk, the correct icon will be displayed.

You can still double-click on a High Sierra file, and it will open correctly.

If a High Sierra disc has been pressed without the Apple extensions, all files on the disc are considered to be of type 'TEXT' and creator 'hscd'. Type 'TEXT' is a generic type which can be read successfully by many Macintosh applications.

Chapter 5. Legal issues

The great thing about CD-ROM is that it stores so much data. You might be tempted to put out a CD with everything you can think of in the way of data. For instance, you may have a collection of digitized sounds that you've downloaded from bulletin boards across the nation. Or you may have recently bought a scanner, and want the world to see the results of the hours of scanning you've done recently. How do you know if what you're about to press is legal? What's separating you from the courthouse besides blind luck?

This section gives you some hints and tips on how to avoid legal hassles with the data you put onto a CD-ROM. I'm not a lawyer, and I don't even play one on TV; if you've got any really thorny issues, or if you're unclear about a point I've mentioned here, go talk to a real lawyer and get your questions cleared up.

Copyright protects the expression of ideas. Copyright does not protect the ideas itself; if you have a piece of code that demonstrates the *if* statement in Pascal, it can't be copyrighted. If you need enough originality to so that some "authorship" exists. Whenever you express an idea, you can potentially have a work that can be copyrighted.

Copyright automatically comes into being when the work is created. There is more to be done in order to have a legally enforceable copyright (you register with the Copyright Office using an official form, and defend the copyright against violation) but in general, copyright exists unless given up. A legally enforceable copyright lasts for 50 years after the death of the author, or 75 years from publication, or 100 years from creation, depending upon the type of work.

Films and videotapes, sound tracks for movies, and pictures in magazines all can be copyrighted, and usually are. Copyright does not end when you transfer an image into your computer; the picture is still copyrighted. Copyright does not end when you digitize a sound from a popular movie into your computer; the sound is still protected by the copyright of the movie.

This means you should be *extremely* careful of how you acquire the artwork, pictures, and sounds you use in creating a CD-ROM. For instance, when Apple created the Apple CD-ROM Explorer disc that you receive when you purchase an AppleCD SC drive, the following steps were taken to make sure that copyright law was not broken:

- Apple had artists create all of the drawings. As work for hire, the drawings belong to Apple for 75 years from publication.
- All the sounds included on the disc were either works of art newly created (again as works for hire, Apple owns the copyright) or explicitly licensed from the owner.
- Digitized pictures of the planets (used the in Star Tour) were acquired from NASA, which is part of the U.S. government. In general, works produced by the U.S. government are in the public domain (but check to make sure!)
- Everything was checked by Apple's legal department, to make sure.

Many good discussions of copyright and trademark law are available. As an example, Nolo Press, 950 Parker Street, Berkeley, California 94710 sells several good legal books aimed at the general public. Of course, whenever you have any doubt, you should check with a lawyer.

Chapter 6. International Considerations

Why worry about localization?

The primary reason to worry about the international market is that you can make more money. Apple derives a large percent of its total revenues from non-USA sales, and we expect this percentage to only increase. Apple now localizes their products in 30 different distinct language/culture formats.

Most of the comments in this section will be of interest to the programmer or graphic artist creating the look and feel of your CD ROM.

Thank you, Mac...the Use of Resources and the Toolbox

The Macintosh was designed to be an international machine. To this end, Apple provides resources and a fairly extensive set of routines to aid in localization of programs. If you use the tools provided to you, your programs can be easily translated from English to a foreign language.

Apple designed resources to contain data used by your program that could be changed without recompilation. Make sure that all of your program strings are contained in resources.

The Macintosh operating system contains routines to correctly sort strings regardless of the language. Be sure and use these routines instead of some library package that comes with your favorite compiler.

Plan on having your program's resources translated by a non-technical person; sometimes, your foreign distributor may not be able to find a programmer who is fluent in both languages. This means avoiding text in a PICT resource (you have to redraw PICTs, and that can be difficult for the naive user.)

Let my ASCII go...

If you assume that everybody uses ASCII, you are sure to have trouble when you try to localize your product to a new country. ASCII stands for the *American* Standard Code for Information Interchange. Other European countries use their own alphabet sets which, like ASCII, are based on an international standard called ISO 646. In countries where the alphabet isn't based on Roman characters (such as Arabic or Japanese), there are other ISO standards.

As an example, Japanese is written using a two-byte character set; if you try to program your own comparisons assuming one-byte characters, you'll fail. Japanese actually has three different encodings for the language; a romanized version using ISO 646, a de-facto standard for MS-DOS and Macintosh computers called shift-JIS, and JIS X0212.

Swedish contains three characters not found in English; **Å**, **Ä**, and **Ö**. Alphabetically, these characters come at the *end* of the Swedish alphabet, after **Z**. If you try to ignore the diacritical marks and sort words beginning with **Å** as if they began with **A**, you'll confuse a Swede.

There are several character sets in wide use for certain languages. For instance, in order to keep to the use of a seven bit character set, German has the character **ß** where the character **[** is in ASCII.

Why don't they say "bread" just like everyone else?

English isn't the only language in wide use in the world. Don't depend upon characteristics of the

English language when working overseas.

In Spanish, *chocolate* sorts after *comida*, because *ch* is considered to be a single distinct character. Sentences that ask a question always begin with an inverted question mark, ¿. You can't tell if a byte is a printable character by checking if it's between **A** and **Z**. The Swedish character **Å** is after **Z** in the alphabet.

Don't depend upon the length of a string. In a language such as Dutch or German, typical prompts, help, and alert messages expand by at least 30 per cent when translated.

Use the Macintosh routines *IUDateString*, *IUTimeString*, *IUMetric*, *IUCompString*, *IUEqualString*, and *IUMagString* all help relieve the problems associated with foreign languages. Be sure and read the *International Utilities* and *Script Manager* chapters of *Inside Macintosh*.

You mean football isn't played with shoulder pads and helmets?

Yes, it's true. American culture, while somewhat pervasive, hasn't replaced every other culture in the world. You should be careful with your choice of icons, pictures, and data so that your choices can be understood overseas.

To state an obvious example, unless you are shipping your product to Burma[1], you should show all measurements in metric measures (kilograms, meters) and not Imperial measures (pounds, feet).

Some countries display time using a 24 hour clock. Some zones of some countries are not an exact number of hours different from Greenwich Mean Time. For example, the middle of Australia is 13 1/2 hours from GMT.

Most of Europe uses the comma as the separator where Americans use decimal point. Instead of using a comma between hundreds in large numbers, they use period. For instance, one thousand two hundred and thirty-four point fifty-six is 1.234,56 rather than 1,234.56. Some currencies, such as the Italian lira, require more digits of precision for their money.

Don't choose images or examples that needlessly offend. For instance, most Arabic cultures are offended by pictures of women in bikinis, or by pictures involving obvious alcoholic beverages. Telephones and mailboxes look quite different in other parts of the world. Envelopes are not addressed the same everywhere; in Russia, the address starts with the equivalent of our State, and works down to the name of the person, which is the last line of the address. Some countries give postal codes before the city name, and some give the postal code after the city name.

You may have data which is considered harmless in the United States, but protected by law in another country. Many countries hold economic or political data to be sensitive and restricted. Be sure that the data you are publishing can be used in the country for which it's intended.

1. Burma is one of the few countries besides the United States of America that hasn't converted over to metric measurements.

Chapter 7. Apple Extensions to ISO 9660[2]

Version 2.3

Bryan Atsatt & Brian Bechtel

Who Should Read This Document

- Apple Developers working with ISO 9660
- Publishers of authoring tools for ISO 9660 discs
- Publishers of ISO 9660 discs
- Publishers of ISO 9660 receiving system software

Release History

1.0	7/7/88	Initial release.
1.1	7/14/88	Removed reference to Macintosh in footnote. Modified note in ProDOS Filename Transformation section to require that Volume Identifier be transformed.
1.2	7/22/88	Changed Protocol Identifier to all uppercase.
1.3	8/2/88	Added SystemUseID 06 for Macintosh Finder flags.
1.4	8/13/88	Corrected location of RecordPad field so comes before the SystemUse field. Added padding fields to SystemUseID's 2-5.
2.0	3/6/89	To accommodate other receiving systems using the SystemUse field, the identification has been slightly modified. A new signature has been added, with some unused fields in the Macintosh extensions removed.
2.1	4/6/89	Described when the Protocol Identifier is required and how to use the extensions on CD-ROM XA discs.
2.2	5/30/89	Corrected SystemUse format description for HFS (SystemUseID 02).
2.3	7/21/89	Added warning about Macintosh versions and what they support. Put back in description of old AppleSignature, for reference.

Introduction

It may be desirable to create an ISO 9660 CD ROM containing HFS and/or ProDOS files in order to benefit by the storage capacity and distribution cost savings of CD ROM, and the interchange advantages of ISO 9660. However, both the HFS and ProDOS file systems require information that the ISO 9660 file system does not support: ProDOS requires a file type and an auxiliary file type, while HFS requires a file type, a file creator, and file attributes.

This document defines a protocol which extends the ISO 9660 specification to include HFS and ProDOS specific information, without corrupting the ISO 9660 structures. Discs created using the protocol are valid ISO 9660 discs, and should not behave differently on non-Apple receiving systems.

In addition to preserving file specific information, the protocol defines a mechanism for preserving filenames across the ProDOS → ISO 9660 → ProDOS translation. The protocol was designed to solve existing compatibility problems as well as allow for future expansion. It uses the SystemIdentifier field in the Primary Volume Descriptor for global information, and the SystemUse field in the directory record for file specific information.

---

2. ISO 9660 is the international file system standard, based on the "High Sierra" standard, for CD ROM. The protocol defined in this document is intended for the ISO 9660 file system; however, it is also supported for the High Sierra file system.

The Protocol Identifier

Location: SystemIdentifier field in the Primary Volume Descriptor
Size: 32 bytes
Contents: "APPLE COMPUTER, INC., TYPE: " followed by a long word type. In hex:

```
41 50 50 4C 45 20 43 4F 4D 50 55 54 45 52 2C 20 49 4E 43 2E 2C 20 54 59 50
45 3A 20 3x 3x 3x 3x
```

Type: The last four bytes of the identifier contain nibble encoded type information. Nibble encoding is necessary in order to guarantee that these are legal ISO 9660 "a-characters" (i.e. printable characters). The type bytes are numbered 0-3; type(0) is the byte following the space ($20). The bits of each type byte are numbered 0-7, 0 being the least significant. The type bytes are defined as follows:

type(0):	bit 0	1 = Perform ProDOS filename transformation (described later)
	bit 1-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(1):	bit 0-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(2):	bit 0-3	reserved, must be zero
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0
type(3):	bit 0-3	Apple Extensions version number (2 indicates the current version, 1 indicates the previous version)
	bit 4	must be 1
	bit 5	must be 1
	bit 6	must be 0
	bit 7	must be 0

The identifier is considered valid if the first 28 bytes match. The identifier is required only for global information; it is not currently required for the Macintosh or for the //GS if the ProDOS filename transformation is not needed under GS/OS.

The Directory Record SystemUse Field

Directory records in the ISO 9660 specification have the following format:

byte	DirectoryRcdLength
byte	XARlength
struct	ExtentLocation
struct	DataLength
struct	RecordingDateTime
byte	FileFlags
byte	FileUnitSize
byte	InterleaveGapSize
long	VolumeSequenceNum
byte	FileNameLength
char	FileName[FileNameLength]
byte	RecordPad (if necessary)
char	SystemUse[SystemUseLength]
byte	RecordPad (if necessary)

The RecordPad field is present only if needed to make DirectoryRcdLength an even number. If present, the RecordPad field *must* be zero ($00). The SystemUse field is an optional field under ISO 9660; it is defined for our use below.

The SystemUse field, when present, must begin with a signature word , followed by a one byte length and a one byte SystemUseID, followed by file system specific information. This structure may be repeated multiple times for multiple file systems; however, the total length of the SystemUse area (SystemUseLength) *must* be an even number. The signature word allows a receiving system to ensure that it can interpret the following data correctly, and the SystemUseID determines the type and format of the information which follows.

CD-ROM XA discs must be handled as a special-case in that the signature word, if present, will begin at byte 14 of the SystemUse field rather than at byte 0.

There are two AppleSignatures. The preferred AppleSignature is defined as "AA" ($41 41). The old AppleSignature, used for a previous version of this document, was defined as "BA" ($42 41). This version did not contain a length byte, and hence made sharing the System Use field with other operating systems impossible. The old AppleSignature is the only one supported by the Macintosh AppleCD SC software version 2.0. Disks pressed using the old format are still supported, but we recommend that new disks be pressed using only the current format.

Receiving systems must perform a simple calculation to determine if the SystemUse field is present in any given directory record. It is present if:

DirectoryRcdLength - FileNameLength > 34

Receiving systems should first verify that the SystemUse field is present (making sure to account for the possibility of a RecordPad field), then check for the AppleSignature before interpreting the SystemUseID.

The Current Version

The SystemUseID is defined as follows for an AppleSignature of "AA". This is the version of the Apple Extensions that is supported by future versions of the Macintosh AppleCD SC software.

SystemUseID	Definition
$00	reserved.
$01	ProDOS file_type and aux_type follow.
$02	HFS fileType, fileCreator, finder flags
$03-FF	reserved.

The following tables define in detail the data formats for each SystemUseID. The MSB-LSB notation ("MSB" = Most Significant Byte, "LSB" = Least Significant Byte) means that the MSB occupies the lowest address, while LSB-MSB means that the LSB occupies the lowest address.

SystemUseID 01, ProDOS:

SystemUse offset	Contents
$00-01	$41 41 (AppleSignature)
$02	SystemUse Extension length ($07 for this ID, includes AppleSignature bytes)
$03	$01 (SystemUseID)
$04	ProDOS file type
$05-06	ProDOS aux type (LSB-MSB)

Note that a padding byte must be added if this ID is the only extension added to the directory record.

SystemUseID 02, HFS:

SystemUse offset	Contents
$00-$01	$41 $41 (AppleSignature)
$02	SystemUse Extension length ($0E for this ID, includes AppleSignature bytes)
$03	$02 (SystemUseID)
$04-$07	HFS fileType (MSB-LSB)
$08-$0B	HFS fileCreator (MSB-LSB)
$0C-$0D	HFS finder flags (MSB-LSB)

The authoring software can simply copy the finder flags as retrieved by the HFS call PBGetFInfo. Only bits 5 (always switchLaunch), 12 (system file), 13 (bundle bit), and 15 (locked) are used. All other bits are either ignored or set due to internal workings of the file system translator. See Macintosh technical note #40 for more details about the finder flags.

The Previous Version

The SystemUseID is defined as follows for an AppleSignature of "BA". This is the version of the Apple Extensions that is supported by the Macintosh AppleCD SC software version 2.0.

SystemUseID	Definition
$00	reserved.
$01	ProDOS file_type and aux_type follow.
$02-05	obsolete versions of HFS; do not use
$06	HFS fileType, fileCreator, finder flags
$07-FF	reserved.

The following tables define in detail the data formats for each SystemUseID. The MSB-LSB notation ("MSB" = Most Significant Byte, "LSB" = Least Significant Byte) means that the MSB occupies the lowest address, while LSB-MSB means that the LSB occupies the lowest address.

SystemUseID 01, ProDOS:	SystemUse offset	Contents
	$00-01	$42 41 (AppleSignature)
	$02	$01 (SystemUseID)
	$03	ProDOS file type
	$04-05	ProDOS aux type (LSB-MSB)

SystemUseID 06, HFS:	SystemUse offset	Contents
	$00-$01	$42 $41 (AppleSignature)
	$02	$06 (SystemUseID)
	$03-$06	HFS fileType (MSB-LSB)
	$07-$0A	HFS fileCreator (MSB-LSB)
	$0B-$0C	HFS finder flags (MSB-LSB)

The authoring software can simply copy the finder flags as retrieved by the HFS call PBGetFInfo. Only bits 5 (always switchLaunch), 12 (system file), 13 (bundle bit), and 15 (locked) are used. All other bits are either ignored or set due to internal workings of the file system translator. See Macintosh technical note #40 for more details about the finder flags.

The Extension to ISO 9660

This section describes the extension to ISO 9660 which allows multiple users of the "System Use" field. All references are to ISO 9660, First edition, 1988-04-15.

Section 9.1.13 System Use [PB (LEN_DR - LEN_SU + 1) to LEN_DR] shall be replaced as follows:

This field shall be optional. If present, this field shall be reserved for system use. If necessary, so that the Directory Record comprises an even number of bytes, a (00) byte shall be added to terminate this field.

If this field is present, it shall be broken up into a series of System Use Extensions. There can be more than one System Use Extension for a given directory record. A System Use Extension shall have the following format:

BP 1 : byte 1 of the signature. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1.

BP 2 : byte 2 of the signature. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1.

BP 3 (LEN_SE): This field shall specify as an 8-bit number the length in bytes of this System Use Extension. This field shall contain an 8-bit number. This field shall be recorded according to 7.1.1. This field shall include the length of the two signature bytes preceding this byte.

BP 4 to LEN_SE : this field shall be reserved for system use. Its content is not specified by this International Standard.

Multiple System Use Extension fields may exist for a given directory record, subject only to the limitation that the total length of a directory record must be able to be recorded in the 8-bit field defined in section 9.1.1.

NOTE: For CD-ROM XA discs, the Byte Positions listed above must be adjusted upwards by 14 to account for XA's 14 fixed length system use field.

The ProDOS Filename Transformation

This section defines a mechanism which preserves ProDOS filename syntax. Legal filenames in ProDOS differ from legal filenames under ISO 9660:

- ProDOS filenames allow multiple periods; ISO 9660 does not.
- ISO 9660 requires that the two separators "." and ";" exist and that the ";" be followed by a version number. This requirement is for files only, not directories.

This requires a transformation of some sort. Fortunately, this specific combination of file systems allows us to define a reversible transformation which the authoring tool and receiving system can use to hide the fact that a transformation has occurred.

When creating an ISO 9660 disc from ProDOS source files, the authoring tool must perform the following transformation on *all* filenames:

- Replace any periods (".", $2E) with underscore ("_", $5F).
- If the filename refers to a directory, the transformation is finished.
- If the filename refers to a file, append the characters ".;1" ($2E, 3B, 31).

Examples:

ProDOS Filename	Type	ISO 9660 Filename
PRODOS	file	PRODOS.;1
BASIC.SYSTEM	file	BASIC_SYSTEM.;1
SYSTEM	directory	SYSTEM
DESK.ACCS	directory	DESK_ACCS
START.GS.OS	file	START_GS_OS.;1

NOTE:The Volume Identifier in the Primary Volume Descriptor is a filename, and, therefore, must also be transformed. It must be transformed as if it is a directory *name.*

When the ProDOS Transformation bit is set in the Protocol Identifier, the receiving system can handle the necessary conversions such that the *original* ProDOS filenames can be used to refer to all files and directories on the volume:

- Before searching the disc for a given pathname, perform the transformation described above.
- Before returning any names found, reverse the transformation above:
  - Strip ".;1" if present.
  - Replace any underscores ("_", $5F) with periods (".", $2E).

This process must be done for every file and directory on disc.

Guidelines For Transforming HFS Filenames

Since most HFS filenames are illegal in the ISO 9660 specification, some transformation will be required when creating an ISO 9660 disc with HFS files. No reversible transformation is possible without degrading performance; therefore, we can only define guidelines for publishers and authoring tool publishers to follow when performing the transformation:

- convert all lowercase characters to uppercase.
- replace all illegal characters, including periods, with underscore ("_", $5F).
- if the filename must be shortened, truncate the rightmost characters.
- if the filename refers to a file, append the characters ".;1" ($2E, 3B, 31).

Following these guidelines will result in more consistent discs.

ISO 9660 Associated Files

Associated files are exactly analogous to resource forks. Though the format of associated files is clear in the ISO 9660 specification, we would like to restate it here:

> An associated file is defined as having the associated bit set in the file flags byte of the directory record. It has exactly the same file identifier as its counterpart, and resides *immediately before* its counterpart in the directory. The associated file is treated as the resource fork, its counterpart is treated as the data fork of the file.
>
> For example, if a file "FOO.;1" has an associated file, there will be two adjacent directory records named "FOO.;1"; the first one (the resource fork) will have the associated bit set, the second one (the data fork) will have the associated bit clear.

The Macintosh File System

The Apple Macintosh file system is named "HFS", for "Hierarchical File System." This file system is similar to both MS-DOS and Unix in supporting subdirectories, where files may be logically grouped together. Subdirectories on the Macintosh are called folders.

When the Macintosh was first introduced, a flat file system called "MFS" for "Macintosh File System" was used. The MFS file system did not adequately provide for high capacity storage media such as hard disks and CD-ROM, so the HFS format was introduced in its place. MFS format is now generally used only on 400K, single sided floppy disks. HFS is compatible with MFS.

Macintosh file and volume names are different from both Unix and MS-DOS in that they allow any character except the colon ":" as a valid file name character. This means that a name such as "My payroll file" is perfectly acceptable on the Macintosh. There is no concept of a file name extension, such as is found in the MS-DOS or High Sierra formats. HFS does not distinguish between upper and lower case names; the names "forecast", "Forecast", and "FoReCaSt" all refer to the same file. Volume names may have a maximum of 27 characters, and file names may have a maximum of 31 characters. To specify a file name in a folder (subdirectory), you give the folder name, followed by a colon, followed by the file name. For example, if you have the file "My payroll file" in the folder "business", you could specify the file as "business:My payroll file". To specify a file in the folder immediately above the current folder, you may use the shortcut name of two colons, "::".

An area where HFS is different from MS-DOS or Unix is in the concept of file forks. A file has two forks; a resource fork and a data fork. The data fork is used for the same purpose as a "file" in MS-DOS or Unix; it contains information in an unstructured format. The data fork is used by an application to store the contents of the document.

The resource fork of a file contains Macintosh resources. Resources are data in a special format; Apple Computer has defined many "famous" types of resources, and we have provided facilities so that programmers may define their own custom resource types. The resource fork of a file is accessed using special program calls, from a part of the Macintosh operating system known as the Resource Manager. All resources are labeled with a resource type of four characters. Some examples of resources are: 'CODE', for executable code; 'ICON' for a 16 bit by 16 bit icon; 'ICN#', a slightly different icon resource used by the Finder to display an unique icon for a file; 'MENU' for the menus found at the top of every Macintosh application; and 'STR ', for strings used within a program. There are many other types of resources as well.

Resources provide the Macintosh with a flexibility not found in other operating systems. For instance, to "localize" a well-written application into another language, all that needs to be changed are the resources that are language specific (usually the 'MENU' and 'STR ' resources.) Apple provides developers with several tools and resource editors that assist in modifying resources.

As well as the resource and data forks, additional information is kept about each file. This information is used by the Finder to display the position, attributes, and icon of a file. The additional information consists of a file type of four characters (such as 'TEXT' for plain text or unknown type documents, 'APPL' for applications, 'MACA' for MacWrite documents, and so forth), a file creator unique to each application (such as 'MWRT', 'EXCL', 'FNDR') and file attributes (such as invisible, system file, or locked against accidental deletion.) The additional information also includes a mapping of file creator to appropriate application, so that a Macintosh user may select a document and open it, and the appropriate application will be invoked and passed the document name.

Problems with High Sierra Support on the Macintosh

Apple supports both High Sierra formats through the use of a feature in the Macintosh file system called the "external file system" hook. This is a special low-memory global which contains a pointer to a list of external file system handlers. For High Sierra support, Apple has written a new set of routines contained in a file called "Foreign File Access". This file, plus the files "High Sierra File Access" and "ISO 9660 File Access" combine to provide complete support for the High Sierra formats. Apple's High Sierra software supports level 2 interchange, according to section 10.2 of the ISO 9660 specification. This means that Apple does not yet support interleaved files or multi-disc volumes.

The Apple extensions do not fully correct deficiencies in the High Sierra specification; some issues are handled by the High Sierra File Access software. For instance, in HFS, the Finder keeps track of the position of a file on the desktop or in a folder by using a special field inside HFS; under High Sierra, a file's position is computed when the folder is opened, and can not be changed.

Because of some deficiencies in the original design of the Finder, it is not possible to have the correct icons display for a file on a High Sierra disc. This is because the Finder does not actually ask for the icon of a file; rather, it assumes the existence of a *desktop database* which contains these mappings, and makes a special call, giving only the file creator and type. The software to provide this information was designed to be very HFS specific. Apple Computer will address this defect in a future release of the operating system. Currently, even if a file's icon bitmap is defined in the Apple extensions, it will not be used by the Finder. All files that reside on a High Sierra disc will display a generic icon. If such a file is copied to a hard disk, the correct icon will be displayed. You can still

double-click on a High Sierra file, and it will open correctly.

If a High Sierra disc has been pressed without the Apple extensions, all files on the disc are considered to be of type 'TEXT' and creator 'hscd'. Type 'TEXT' is a generic type which can be read successfully by many Macintosh applications.

There has been some confusion about padding bytes in a directory record which also contains Apple extensions. There are two possible places in a directory record where a padding field may occur: after the File Identifier, and at the end of the directory record. If no Apple extensions exist for a file, then the two locations are identical, and the rule stated in section 9.1.12 applies: "...(a padding field) shall be present ...only if the number in the Length of the File Identifier is an even number." If Apple extensions exist, then section 9.1.13 *also* applies: "If necessary, so that the Directory Record comprises an even number of bytes, a (00) byte shall be added to terminate this field." (i.e. the System Use field used for Apple extensions.)

Sample Code for Retrieving Macintosh File Information

This code is an example of how to access the Macintosh file type, file creator, and finder flag information. The code is written in the C language.

```
/*****************************************************************************
 *
 *  Function:         GetFileInfo
 *
 *  Purpose:      get lengths of file rsrc and data forks, type, creator,
 *                and finder flags
 *
 *  Returns:      OSErr
 *                noErr, unless PBGetFInfo has an error.
 *
 *  Side Effects: rsrcLength, dataLength, fType, fCreator, flags are
 *                    updated with correct values for the file requested
 *
 *  Description: call PBGetFInfo() and return its results.  This routine
 *                will only work if the path name to the file on the Mac
 *                can fit in 255 characters (the length of a pascal
 *                string)  See Inside Macintosh, Volume 2, page II-115
 *                and Mac Tech Note #40 for more information.
 *
 *****************************************************************************/
OSErr
GetFileInfo(name, vRefNum, rsrcLength, dataLength, fType, fCreator, flags)
StringPtr    name;
short        vRefNum;
long     *rsrcLength;
long     *dataLength;
OSType       *fType;
OSType       *fCreator;
short        *flags;
{
    HParamBlockRec   io;
    OSErr            result;

    io.fileParam.ioCompletion = 0L;
    io.fileParam.ioNamePtr = name;
    io.fileParam.ioVRefNum = vRefNum;
    io.fileParam.ioFVersNum = 0;
    io.fileParam.ioFDirIndex = 0;
    result = PBGetFInfo(&io, false);
    if (result == noErr)
    {
        *rsrcLength = io.fileParam.ioFlRLgLen;
        *dataLength = io.fileParam.ioFlLgLen;
        *fType = io.fileParam.ioFlFndrInfo.fdType;
        *fCreator = io.fileParam.ioFlFndrInfo.fdCreator;
        *flags = io.fileParam.ioFlFndrInfo.fdFlags;
    }
    else
    {
        *rsrcLength = 0L;
        *dataLength = 0L;
        *fType = 0L;
        *fCreator = 0L;
        *flags = 0;
    }
    return result;
}
```

Chapter 8. Macintosh Audio Calls

Audio compact discs and CD-ROM discs conform to two standards, called the Yellow Book and the Red Book. The Yellow book specifies audio standards; the Red book specifies additional standards for CD-ROM. In order to use call the Macintosh driver, you need to understand the possible addressing formats for audio blocks on a CD. The three addressing formats are:

1) logical block
2) absolute minute, second and block
3) track number.

An audio CD contains up to 74 minutes of sound (By decreasing the inter-track gap and other tricks, you can put more than 74 minutes of sound on a CD. Doing this is only marginally within the standard. Such CDs may not be playable on all drives.). These minutes are divided into 60 seconds. Each second contains 75 blocks. Each block contains 98 frames. Each frame contains 24 bytes of data. The music on a CD is divided up into tracks; you can have a maximum of 99 tracks on a CD.

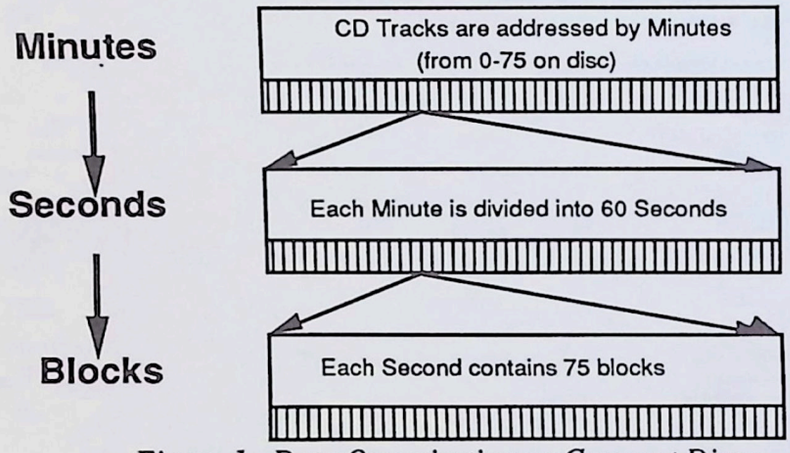


Figure 1. Data Organization on Compact Discs

You can address sound down to the block level; that is, down to the 1/75th of a second. You can address sound by specifying an absolute block number (e.g. start playing at absolute block 1,234,567 from the start of the disc), or by absolute minute, second, and block number (e.g. start playing at minute 42, second 30, block 15 on the disc) or by logical track number (e.g. start playing at track 2.)

The driver requires that the logical track number or absolute minute second block numbers be specified in BCD (e.g. the decimal number 12 is stored as hex 0x0102, not as 0x000C.)

Many calls return a control field. This field describes the format of the current track, and has the following values:

bits 3210	function
00x0	2 audio channels without preemphasis
00x1	2 audio channels with preemphasis
10x0	4 audio channels without preemphasis
10x1	4 audio channels with preemphasis
01x0	data track
01x1	reserved
11xx	reserved
xx0x	digital copy prohibited
xx1x	digital copy permitted

Many calls return a play mode. This field describes how to play the audio track. This field has the following values:

bits 3210	effect
0000	muting on (no audio)
0001	right channel through right channel only
0010	left channel through right channel only
0011	left and right channel through right channel only
0100	right channel through left channel only
0101	right channel through left and right channel
0110	right channel through left channel, left channel through right channel
0111	right channel through left channel, left and right channel through right channel
1000	left channel through left channel only
1001	left channel through left channel, right channel through right channel (stereo)
1010	left channel through left and right channel
1011	left channel through left channel, left and right channel through right channel
1100	left and right channel through left channel
1101	left and right channel through left channel, right channel through right channel
1110	left and right channel through left channel, left channel through right channel
1111	left and right channel through left channel, left and right channel through right channel (monaural)

In the absolute minute-second-block address, the ranges are as follows:

Minutes	00 to 99 (effectively, from 00 to 75)
Seconds	00 to 59
Blocks	00 to 74

To issue a call to the driver, you can use either the high-level or low-level driver calls described in the Device Manager chapter of *Inside Macintosh* volume 2 starting at page II-175.

The high level calls require that you pass in three parameters; the driver reference number, the csCode indicating which call is being made, and a pointer to the csParam area containing control information specific to that call. For example, to issue a type 1 ReadTOC call you could issue the following commands (this example is in MPW C):

```
#include <Files.h>

/* set up somewhere else by an OpenDriver() call */
short         drvRefNum;

/*
** GetFirstAndLastTrackNumber
**      requires two parameters:
**      1) a pointer to a short which will hold the first track number in BCD
**      2) a pointer to a short which will hold the last track number in BCD
**
**      fills in these two pointers with the first and last track numbers, or
**      zero if an error occurred.  Returns whatever was the driver return code.
*/
OSErr
GetFirstAndLastTrackNumber(firstTrack, lastTrack)
short *firstTrack;
short *lastTrack;
{
        OSErr  result;              /* for storing result of Control() call */
        char   csParam[22];

        csParam[0] = 0;
        csParam[1] = 1;
        result = Control(drvRefNum, 100, &csParam);
        if (result == noErr)
        {
                *firstTrack = csParam[0];
                *lastTrack = csParam[1];
        }
        else
        {
                *firstTrack = 0;
                *lastTrack = 0;
        }
        return(result);
}
```

The low-level calls require that you pass in two parameters; a pointer to a parameter block and a flag indicating synchronous or asynchronous completion. For now, the driver always is synchronous, so always pass false as the second parameter. You must fill in the parameter block yourself. You may find it easier to define your own custom parameter block variants for the various calls. Some example MPW C code to get audio status information is as follows:

```
#include <Devices.h>

short  drvRefNum;          /* set up somewhere else */

/* this is a custom version of the CntrlParam parameter
** block defined on page II-181 and II-183 or in
** Devices.h
*/
typedef struct {
      QElemPtr     qLink;
      int          qType;
      int          ioTrap;
      Ptr          ioCmdAddr;
      ProcPtr            ioCompletion;
      OsErr        ioResult;
      StringPtr    ioNamePtr;
      int          ioVRefNum;
      int          ioRefNum;
      int          csCode;
/* Everything below this replaces: short csParam[11] */
      unsigned char status;
      unsigned char play;
      unsigned char control;
      unsigned char minute;
      unsigned char second;
      unsigned char block;
      char   unused[16];         /* for the rest */
} CDCntrlParam;

/*
** GetAudioStuff
**     requires one input, the driver reference number that we
**     got by calling PBOpen().
**     fills in six parameters with appropriate information
**     from the driver call.
**     We're assuming that only one CD player is attached.
**     To generalize this code, fill in the ioVRefNum dynamically.
*/
OSErr
GetAudioStuff(status, play, control, minute, second, block)
unsigned char *status, *play, *control, *minute, *second, *block;
{
      OSErr  result;
      CDCntrlParam  myPB;

      myPB.ioCompletion = 0;
      myPB.ioVRefNum = 1;        /* we're assuming the 1 volume */
      myPB.ioRefNum = drvRefNum; /* determined elsewhere */
      myPB.csCode = 107;

      result = PBControl(&myPB, false);
      if (result == noErr)
      {
            *status = myPB.status;
            *play = myPB.play;
            *control = myPB.control;
            *minute = myPB.minute;
            *second = myPB.second;
            *block = myPB.block;
      }
      return(result);
}
```

ReadTOC (100) **Read Table of Contents**

Description: ReadTOC returns the table-of-contents information of a particular CD. There are three types of ReadTOC calls. They are distinguished by the contents of the word at csParam+0.

Input:

csParam+0 (word)	Type of ReadTOC call:
0x0001	Request the first and last user tracks.
0x0002	Request the lead-out (end of disc) address in absolute minute-second-block format.
0x0003	Request track starting addresses for user-specified range of tracks.

For type 1 and type 2 calls, the information requested is returned directly in the csParam field. For type 3 calls, the user must allocate a data buffer for the data returned by the driver.

The following additional parameters are needed for type 3 calls:

csParam+2 (long)	address of data buffer
csParam+6 (word)	length of data buffer
csParam+8 (byte)	starting track number in BCD

The driver calculates the number of tracks from the length of the data buffer. If you allocate too little or too much space, undefined errors may result (you'll stomp on memory.)

Output:

For Type 1 calls:

csParam+0 (byte)	First track number in BCD
csParam+1 (byte)	Last track number in BCD

For Type 2 calls:

csParam+0 (byte)	lead out (end of disc) address, minutes, in BCD
csParam+1 (byte)	lead out (end of disc) address, seconds, in BCD
csParam+2 (byte)	lead out (end of disc) address, blocks, in BCD

For Type 3 calls:

Four bytes of data will be returned in the buffer specified, for each track in the following format:

Byte	Content
0	Control field of specified track (see page 1)
1	track start, absolute minutes, in BCD
2	track start, absolute seconds, in BCD
3	track start, absolute blocks, in BCD

ReadQ (101) **Read Q Subchannel**

Description: ReadQ returns the position information stored in the Q subchannel of the disc.

Input: none

Output:

csParam+0 (byte)	Control field (see page 1)
csParam+1 (byte)	track number in BCD (range 1 to 99)
csParam+2 (byte)	Index
csParam+3 (byte)	location, minutes, from start of track, in BCD
csParam+4 (byte)	location, seconds, from start of track, in BCD
csParam+5 (byte)	location, block, from start of track, in BCD
csParam+6 (byte)	absolute location in minutes, from start of disc, in BCD
csParam+7 (byte)	absolute location in seconds, from start of disc, in BCD
csParam+8 (byte)	absolute location in blocks, from start of disc, in BCD

ReadHead (102) **logical block information**

Description: Given a specific logical block, returns the absolute minute-second-block address and mode.

Input:

csParam+0 (long)	logical block number

Output:

csParam+0 (byte)	absolute minutes, in BCD
csParam+1 (byte)	absolute seconds, in BCD
csParam+2 (byte)	absolute blocks, in BCD
csParam+3 (byte)	Mode

A return code of noErr means that this is a data block. A return code of paramErr means this is an audio block.

ATrkSearch (103) **Audio Track Search**

Description: Positions the optical pickup at the specified audio address. A paramErr is returned if the address specified is not in an audio portion of the disc (either out of bounds or a data track.)

Input:

csParam+0 (word)	0x0000 (search address in logical block format.)
csParam+2 (long)	Type 0: logical block address: most significant byte in byte 2, least significant byte in byte 5 (normal Macintosh order.)
csParam+6 (word)	Play flag 0 = search and pause at this address 1 = search and play according to play mode.
csParam+8 (word)	Play mode. (see page 1)

or

csParam+0 (word)	0x0001 (search address in absolute minute-second-block format.)
csParam+2 (byte)	00
csParam+3 (byte)	absolute minutes in BCD
csParam+4 (byte)	absolute seconds in BCD
csParam+5 (byte)	absolute blocks in BCD
csParam+6 (word)	Play flag 0 = search and pause at this address 1 = search and play according to play mode.
csParam+8 (word)	Play mode. (see page 1)

or

csParam+0 (word)	0x0002 (search address in track number format, in BCD.)
csParam+2 (byte)	00
csParam+3 (byte)	00
csParam+4 (byte)	00
csParam+5 (byte)	track number in BCD
csParam+6 (word)	Play flag 0 = search and pause at this address 1 = search and play according to play mode.
csParam+8 (word)	Play mode. (see page 1)

APlay (104) **Play Audio**

Description: Positions the optical pickup at the specified audio address. A paramErr is returned if the address specified is not in an audio portion of the disc (either out of bounds or a data track.) If you haven't issued an AStop before issuing an APlay then play continues until the end of the disc.

Input:

csParam+0 (word)	0x0000 (play address in logical block format)
csParam+2 (long)	Type 0: logical block address: most significant byte in byte 2, least significant byte in byte 5 (normal Macintosh order.)
csParam+6 (word)	Stop address flag 0 = this is a starting address. Start play at this address. 1 = this is a stopping address. This call is equivalent to an AStop call.
csParam+8 (word)	Play mode. (see page 1)

or

csParam+0 (word)	0x0001 (play address in absolute minute-second-block format)
csParam+2 (byte)	00
csParam+3 (byte)	absolute minutes in BCD
csParam+4 (byte)	absolute seconds in BCD
csParam+5 (byte)	absolute blocks in BCD
csParam+6 (word)	Stop address flag 0 = this is a starting address. Start play at this address. 1 = this is a stopping address. This call is equivalent to an AStop call.
csParam+8 (word)	Play mode. (see page 1)

or

csParam+0 (word)	0x0002 (play address in track number format, in BCD.)
csParam+2 (byte)	00
csParam+3 (byte)	00
csParam+4 (byte)	00
csParam+5 (byte)	track number in BCD
csParam+6 (word)	Stop address flag 0 = this is a starting address. Start play at this address. 1 = this is a stopping address. This call is equivalent to an AStop call.
csParam+8 (word)	Play mode. (see page 1)

APause (105) **Audio Pause/Resume**

Description: Starts or stops the current APlay operation. This call is undefined if a prior APlay or ATrkSearch call has not been made.

Input:

csParam+0 (long)

0x00000001	Enter hold state and mute audio
0x00000000	Release pause and resume play at next block.

AStop (106) **Stop Audio Play**

Description: This control call specifies the audio address where the APlay operation will stop. This is a terminating address; if you issue this call using the track number, play will terminate at the end of this track.

Input:

csParam+0 (word)	0x0000 (stop address in logical block format.)
csParam+2 (long)	Type 0: logical block address: most significant byte in byte 2, least significant byte in byte 5 (normal Macintosh order.)

or

csParam+0 (word)	0x0001 (stop address in absolute minute-second-block format.)
csParam+2 (byte)	00
csParam+3 (byte)	absolute minutes in BCD
csParam+4 (byte)	absolute seconds in BCD
csParam+5 (byte)	absolute blocks in BCD

or

csParam+0 (word)	0x0002 (stop address in track number format, in BCD.)
csParam+2 (byte)	00
csParam+3 (byte)	00
csParam+4 (byte)	00
csParam+5 (byte)	track number in BCD

AStatus (107) **Audio Status**

Description: Returns the status of the current audio block. If the drive is not in a ready condition, ioErr is returned. If this call is issued while on a data block, an ioErr is returned. This call returns the type of play currently being done, the type of track, and the absolute location on disc.

Input: none.

Output:

csParam+0 (byte)	Audio Status 0: audio play in progress 1: audio pause in operation 2: audio muting on 3: audio play operation completed 4: error occurred during audio play 5: not currently playing
csParam+1 (byte)	Play mode. (see page 1)
csParam+2 (byte)	Control field (see page 1)
csParam+3 (byte)	absolute minute in BCD
csParam+4 (byte)	absolute second in BCD
csParam+5 (byte)	absolute block in BCD

AScan (108) **Audio Scan**

Description: This control call starts a fast-forward or fast-reverse scan operation on the disc starting from the specified address.

Input:

csParam+0 (word)	0x0000 (scan address in logical block format.)
csParam+2 (long)	Type 0: logical block address: most significant byte in byte 2, least significant byte in byte 5 (normal Macintosh order.)
csParam+6 (word)	Direction: 0x0000: fast forward 0x0001: fast reverse

or

csParam+0 (word)	0x0001 (scan address in absolute minute-second-block format.)
csParam+2 (byte)	00
csParam+3 (byte)	absolute minutes in BCD
csParam+4 (byte)	absolute seconds in BCD
csParam+5 (byte)	absolute blocks in BCD
csParam+6 (word)	Direction: 0x0000: fast forward 0x0001: fast reverse

or

csParam+0 (word)	0x0002 (scan address in track number format, in BCD.)
csParam+2 (byte)	00
csParam+3 (byte)	00
csParam+4 (byte)	00
csParam+5 (byte)	track number in BCD
csParam+6 (word)	Direction: 0x0000: fast forward 0x0001: fast reverse

DiskStatus (8) **Return Disk Status**

Description: Returns information about a specific drive, such as whether there is a CD in place. This is a status call, not a control call. Use `PBStatus` to get this information.

Input:

none.

Output:

csParam+0:	word	0 (track)
csParam+2:	byte	0x80 (write protected)
csParam+3:	byte	diskInPlace 1 = yes 0 = no
csParam+4:	byte	1 (installed)
csParam+5:	byte	1 (sides)
csParam+6	long	NIL (qLink)
csParam+10:	word	0 (qType)
csParam+12:	word	dQDrive = drive number
csParam+14:	word	dQRefNum = drive reference number
csParam+16:	word	1 (dQFSID)
csParam+18:	byte	0 (twoSideFmt)
csParam+19:	byte	0 (needsFlush)
csParam+20:	byte	0 (diskErrors)

Return codes:

`noErr`	no error.
`badUnitErr`	bad reference number.
`unitEmptyErr`	bad reference number.
`notOpenErr`	driver is not open.

WhoIsThere (97) **Find AppleCD SC drives**

Description: Lets an application know what SCSI IDs are AppleCD SC drives. This is a status call, not a control call. Use `PBStatus` to get this information.

Input:

none.

Output:

csParam+0:	byte	0
csParam+1:	byte	bits corresponding to each of the possible 8 SCSI ids, 0 through 7. SCSI ID 7 is the left most (most significant) bit, SCSI ID 0 is the right most (least significant) bit. For example, SCSI IDs 7 and 1 would be represented as `10000010`.

Return codes:

`noErr`	no error.
`badUnitErr`	bad reference number.
`unitEmptyErr`	bad reference number.
`notOpenErr`	driver is not open.

Chapter 9. Ask Swami ROMmie

Swami ROMmie used to spend his time pushing optical media for the Feds in Washington. Then one day he stared too hard at those neat rainbow patterns on a CD and fell into the infinite void. Now he's adopted an Eastern religion, moved to California, and spends his time meditating on the future of information retrieval somewhere in the Berkeley Hills. Originally, Swami ROMmie was going to be a Zen Master, but he found out that being a SCSI Master was cheaper and easier on the knees. Apple has put the Swami on retainer to answer your questions — we'll make periodic pilgrimages to receive his revelations. May I have the first envelope please...

Dear Swami: I mastered a Macintosh CD-ROM from a hard disk image. It does work, but it takes a long time to mount on the Desktop, and some Macs crash during boot if the CD-ROM is loaded in the drive. What happened — how can I fix my disc and achieve nirvana? — Bootless in Cupertino

Dear Bootless: You've got a combination of two problems. First, your CD-ROM probably contains boot blocks copied from the hard disk. These are the first two logical blocks of the volume: whenever you copy a "blessed" System Folder onto a hard disk these blocks are filled in with boot blocks for that particular configuration. When your Mac starts up, its own boot code looks at each of the SCSI devices, from high number to lower, searching for a device with valid boot blocks. When it finds the first one, it reads the blocks and tries to run the code in them. The problem comes when the system tries to use those copied hard disk boot blocks to read a CD-ROM drive — very bad karma. A particular configuration will only fail if the CD drive is the first (highest numbered) "bootable" device, that's why you don't always see the problem.

To solve the problem, before mastering take a look at the first two bytes in your hard disk volume with a disk editor like Symantec Utilities, ZAP, MacSnoop, or FEdit. If you've got boot blocks there, you'll see an ASCII 'LK' in those first two bytes. Replace them with anything else, like zeros. Now the Mac won't try to boot your CD-ROM. (The 'LK' stands for Larry Kenyon, for trivia fiends.)

Slow loading can happen several ways. One happens if you (or the CD-ROM company) forgot to dismount the hard disk volume before making a copy for mastering. Whenever an HFS disk is mounted, a bit is set in the volume's header, and it's cleared again when you dismount the disk or shut down the Mac normally. (Ever wondered what "You may now shut off your Mac safely" means — now you know!) The purpose of this bit is a "crash flag" — if the Mac dies abnormally, leaving the bit set, at the next boot the Mac will recheck the integrity of the file structure and Desktop of the disk, just in case something got smashed during the crash. So... if you master from a hard disk that has the bit set because you didn't dismount it, the Mac will think your CD-ROM is potentially crashed every time you load it, and will spend a lot of time rechecking EVERYTHING on the CD. Sin no more — dismount all hard disks before mastering.

Another possibility is that the file structure on your hard disk was fragmented or the Desktop file was full of old icons and other clutter. To avoid this problem, remember the words of the

master: "The first step on the path to nirvana is clean layout." This means you should make a backup copy of all of your information, reformat the hard disk, and then copy it all back just before mastering. This will make sure your files are laid out contiguously, without fragments, and the Desktop file will contain only what you need. Formatting will also clean off any "dead file" data that's still recorded on the disk even though the file has been trashed. (Do you really want 2,000 copies of your last performance appraisal circulating around the world? I didn't think so...)

Dear Swami: I built a CD-ROM with HyperCard stacks. It all works, but it takes so long to jump between stacks that you can go out for coffee. My customers are complaining — they're all getting caffeine hangovers. I've prayed to Saint Bill and it didn't work, now what? — Stacked Against the Wall

Dear Stacked: Between stack jumps will always take longer, because HyperCard has to go through the File Manager to find the target stack, and then load its pointer and hash tables. But there are things you can do help out. First, heed my advice to "Bootless" — reformat your disc and copy in fresh files to clean up the file and Desktop structure. If you have a lot of stacks, nest them into folders — long file lists take longer to load. Be smart about it — put related stacks that do lots of cross linking into the same folder. Consider whether some of your stacks could be separate for production purposes, but merged at the last minute for speed. Suggest that your users set a RAM cache, if their machine has enough memory to do that and run HyperCard (2 Meg and up). And TEST, TEST, TEST — the interactions between stack size, HyperCard and cache RAM space, and apparent speed can be very complex.

There's always going to be a difference in speed between intra- and inter-stack jumps, so remember the words of St. Userface: "Predictability can be as important as performance." Don't surprise your users with a long delay. For instance, never have one button that sometimes works quickly and sometimes slowly. You might want to consider overriding the "go" command — change the cursor to a watch for a short jump and something else for a cross stack jump — a coffee cup? (And remember to "pass go" at the end, or you're not going anywhere...)

Dear Swami: I'm about to press my CD-ROM. How can I be sure that the data is organized so that it runs faster than the proverbial nocturnal flying mammal from the nether world? I understand your wisdom about copying everything off the source hard disk and reformatting before copying back, but some of us are not blessed with TWO 600 Megabyte storage devices. — Resources Cut but Still Trying

Dear Trying: To optimally rearrange your data on a single disk, you need a disk optimizer program which will do the job in place. The Swami knows of three commercial disk optimizers:

1) DiskExpress.
2) Symantec Tools for the Macintosh (SUM).
3) PowerOp.

Of the three, I prefer DiskExpress. Let it run over your entire disk, and it will make all the files contiguous again. Warning: this takes a LONG time. Make backups of key files in case of a power failure during the process. While your machine is busy you can go beg for more budget. The Swami finds that taking along a live cobra helps.

Dear Swami: I'm trying to build a HyperCard CD-ROM. When I try to link between stacks I get a cryptic "file system error -37" message box. Worse, now I sometimes get the same error when I'm working on other projects in HyperCard. Save me from the creeping crud, Swami!

Signed, This is Intuitive?

Dear Into-it: This turns out to be a HyperCard problem, not specifically CD related. The Swami was forced to meditate on the image of Saint Bill — he finds that wearing a loud Hawaiian shirt helps. It was revealed to him that HyperCard's error checking is rather weak in the area of folder names. Particularly, the Good Book of Macintosh preaches that no folder name shall be longer than 31 characters. However, HyperCard makes no check that you are being righteous. For instance, try executing the following from the message box after replacing MyHardDisk with the name of a volume you have mounted:

```
go to stack "MyHardDisk:This is much longer than 31 characters:foobar"
```

Voila! -37, right? Now you are halfway to the solution. To understand the creeping plague, go put the same volume & folder name (delete the final "foobar") into the "look for stacks" card in your home stack. Now try

```
go to stack foobar
```

-37 again, right? As long as you have an invalid path in the home stack, you will get a -37 every time HyperCard finds it necessary to use that particular list, that is, every time you don't give it an explicit path to the stack, document or application. You should now find it a trivial matter to detect and delete the item causing your -37 errors.

Dear Swami: I've hooking up my AppleCD SC drive and I'm having all sorts of trouble with it and my other SCSI devices. The flakies have hit. What could be wrong? Have I lead an unclean life? — Doubtful on DeAnza

Dear Doubtful: We have all sinned (at least in our hearts) in this life and past lives, but the Swami finds that most people having trouble with the Apple CD drive haven't terminated their SCSI chain properly. You should have one terminator at the beginning of the chain, and one at the end. If you have an internal hard disk, it is already terminated and is the beginning of the chain. Put the ending terminator IN-LINE with the SCSI cable, not on the other available connector, and all should be well. While the Swami is a SCSI master (it's cheaper than being a Zen master), he finds that one needs all the good karma available when coping with Mac SCSI.

Dear Swami: This bug is driving me nuts! I seem to be getting a "directory not found" error (-120) from my application, but ONLY on High Sierra or ISO 9660 CDs. My code works fine with HFS disks. Is there a bug in Apple's software for these formats, or what? — Bugged on Bandley

Dear Bugged: Ah, grasshopper, as we all know, there are no bugs in Apple code...you are probably saving directory ids as a short (16 bits) in your application, but directory ids are actually longs (32 bits). On most HFS disks, this doesn't matter; you must have many, many files on a disk to get that long a directory. But on High Sierra and ISO 9660 CDs, the directory id is always a long value. Your code is truncating the directory id to 16 bits, and kaboom!

Oh Swami: I had dream in which a lotus flower in full bloom emerged from a stone wall. After just seconds the flower petals closed around a pearl. I could not persuade the flower to open and release the pearl. I need your help in understanding how this dream relates to my search for the truth regarding locked media and user text input in HyperCard. I am creating a stack that will be distributed on a CD. I have the need to prompt the user for some information and would like them to be able to type directly into the card. I am aware that this information will be lost when the card is left. How does one test this? When I set the file to "locked" on the Finder, or set the "cantmodify" of the stack (in the Protect Stack... DLOG) to true (to test and

simulate the CD environment), HyperCard returns the error, "Can't modify this stack". How can I be sure this message will not appear when running off the CD? How does UserModify and CantModify differ? — Stumped at the wall

Dear Stumped: The lotus opens. The stone wall is a chimera. The pearl drops on the floor and rolls under the bed, just out of your reach. The plus 2 dragon hits...(oops, sorry, wrong message.)

To test the effects of running from a CD-ROM, you can try running on a locked floppy disk. A CD-ROM and a locked floppy behave the same as far as Hypercard's "cantmodify" and "usermodify" are concerned. The difference between "usermodify" and "cantmodify" is as follows:

"cantmodify" reflects the actual state of the stack. If cantmodify is true, then either the stack is on read-only media (CD-ROM, locked floppy) or the stack has the finder lock bit set (via Get Info.) You can also set the status of cantmodify by the command

```
set the cantmodify to true
```

Notice that this isn't a stack property, but a Hypercard property. You can't alter the cantmodify of a locked stack.

"usermodify" modifies the behavior of a stack where cantmodify is true. If cantmodify is true and usermodify is false, you can't enter any information into a field. You'll get the message "Can't modify this stack." If cantmodify is true and usermodify is true, you can enter information into a field, but the information goes away when you exit the card. If cantmodify is false, then usermodify has no effect. You can set the usermodify of a stack by the command

```
set the usermodify of this stack to true
```

Notice that usermodify is a stack property, not a Hypercard property.

Until next time, ROMmmmmmm.....

Chapter 10. Operating System Support for ISO 9660

This chapter describes the support provided for ISO 9660 by the five popular operating system implementations. These systems are:

- Microsoft MS-DOS Extensions, for MS-DOS.
- Digital Equipment Corporation's VFS, for VMS.
- Apple Foreign File Access for Macintosh
- Apple IIe ProDOS support routines
- Apple IIgs GS/OS file system translator, for GS/OS (a new IIgs operating system).

All five operating systems:

- support a logical sector size of 2048 bytes only. The ISO 9660 specification allows for any multiple of 2048 bytes.
- do not use the "System Area" field.
- require a Primary Volume Descriptor.
- do not use a Boot Record Descriptor, but allow one to exist on the disc.
- require a Volume Descriptor Terminator.
- allow for 1 root and 7 nested subdirectories, with files in any directory.
- do not allow multi-extent files.

Feature	Microsoft MS-DOS Extensions	Digital VMS	Macintosh ISO 9660 Access	Apple IIGS/OS	Apple IIe Prodos
software version	2.0	1.0	1.0	1.0	1.0
Associated Files	allowed; not shown	supported	supported; used for resource forks	supported; accessible through call to driver	supported; accessible through call to driver
interleaved files	not supported	supported	supported in version 2.0	supported	supported
allowable logical sector sizes	2048	2048	2048	2048	2048
allowable logical block sizes	512, 1024, 2048	512	512, 1024, 2048	512, 1024, 2048	512, 1024, 2048
existence bit	file has "hidden" attribute; not shown during normal directory activity	file has "hidden" attribute; not shown during normal directory activity	file has "invisible" attribute; not shown during normal directory activity	file has "invisible" attribute; not shown during normal directory activity	file has "invisible" attribute; not shown during normal directory activity
protection bit	supported; file is not accessible	supported; file is accessible only to proper group or owner	not supported; file is accessible	not supported; file is accessible	not supported; file is accessible
Extended Attribute Records	not supported	supported; used for group and owner information	recognized and ignored	recognized; available via driver call	recognized; available via driver call
System Area		Not used	Not used		Not used
Supplementary Volume Descriptor	Not used Allows shift-JIS Kanji SVD in specified position	Not Supported	supported in version 2.0	Not used Not Supported	Not Supported
maximum path size	64 characters	255 characters	255 characters	255 characters	255 characters

Index

3M Optical Recording 3
Apple Extensions to ISO 9660 17, 22
associated files 28
Audio control calls
  Audio scan 44
  Pause/Resume 41
  Play 40
  Read table of contents 36
  ReadHead 38
  ReadQ 36
  Status 43
  Stop 42
  Track search 39
Audio status calls
  Disk status 45
  Who is there? 46
CD-ROM EndUser 2
CD-ROM features 1
Characteristics of CD-ROM 4
CIRC 7
Clearing copyright 19
CRC 10
Cross Interleave Reed-Solomon Coding 7
DADC 3
data fork 15
DDRI 2
default Macintosh type & creator for High Sierra CD-ROMs 18
Definition of "resource" 15
Definition of CD-ROM 1
Denon America 3
Description of Macintosh HFS file system 28
Digital Audio Disc Corporation 3
Disctronics 3
ECC 10
EDC 10
EFM 6
File and document icons under High Sierra 17
File and document icons under ISO 9660 17
file attribute 15
file creator 15
file forks 15
file type 15
Foreign File Access 17
HFS 14
High Sierra 10, 13
High Sierra File Access 17
High Sierra versus ISO 9660 14
How long does a copyright last? 19
How to press a CD-ROM 2
How to tell if "Apple Extensions to ISO 9660" are in use 24
ISO 9660 10, 14, 52
ISO 9660 extensions
  AppleSignatures 24
  associated files 28
  detecting extensions 24
  Directory record System Use field 24
  HFS filename transformation 28
  old AppleSignature 24
  ProDOS filename transformation 27
  protocol identifier 23
  system id "AA" 24
  system id "BA" 25
ISO 9660 File Access 17
ISO 9660 format
  associated file 17
  directory records 16
  logical blocks 15
  logical sectors 15
  path table 16
  primary volume descriptor 16
  secondary volume descriptor 16
  volume descriptors 15
Legal issues 19
Low-level Data Encoding 6
Mac audio calls
  Control field format 33
  play mode format 33
  sample C low level call 35
  sampleC high level call 34
Macintosh versus other file systems 14
making your CD run fast 3
MFS 14
Nimbus Information Systems 3
Nolo Press 19
Philips Dupont Optical 3
Physical Media of CD-ROM 5
Problems
  Bad file name error (-37) 48
  CD takes long time to mount 47
  directory not found (-120) 49
  How to optimize a CD-ROM 48
  Hypercard and path names 48
  Hypercard is slow on CD-ROM 48
  Installing AppleCD SC drive 49
  `usermodify` versus `cantmodify` 49
Problems with High Sierra Support on the Macintosh 29
Quick summary: How to Press a CD 2
Red Book 1
resource fork 15
Sample Code
  High level audio calls 34
  Low level audio calls 35
Sample Code for Retrieving Macintosh File Information 31
Subcoding Channels and Blocks 8
Swami ROMmie 47
Table of ISO 9660 support 52
Why do High Sierra CD-ROMs have generic icons? 17
Why worry about localization? 20
Yellow Book 1